Solution for Site Multihoming in a Real IP Environment
draft-shyam-site-multi-43.txt

Abstract

This document provides a solution for Site Multihoming of stub networks in a real IP environment. Each user interface in a customer network may have as many global unicast addresses as many service providers it will be connected with. Users can establish multiple connections through different service providers simultaneously. Customer networks can maintain private address space to communicate within its users. Customer networks can provide IP mobility services as well.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 12, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
1. Introduction

Based on the definition of "multihoming" as stated in RFC3582[1],

"A "multihomed" site is one with more than one transit provider. "Site-multihoming" is the practice of arranging a site to be multihomed."

This is a general solution for site multihoming of stub networks in a real IP world irrespective of the framework supported by the service provider network. The solution is applicable to any customer network that receives globally unique IP addresses for all of its nodes and communicates with the rest of the world without the help of NAT[17]. It is applicable to any version of IP, i.e. IPv4, IPv6 or any new generation of IP that may emerge by removing the drawbacks associated with IPv6[7]. Within a provider assigned address space, each customer network will possess as many global unicast address space as many service providers it gets connected with. So, an user interface of a host may have as many global unicast addresses as many service providers it will be connected with.

Users can maintain multiple connections through multiple service providers simultaneously. A customer network can maintain private IP addresses to communicate within its users. Communication using private IP is restricted to private IP space for the sake of privacy. Customer networks can provide IP mobility support as well.
There are many variants of UNIX systems (as well as real time operating systems) which make use of BSD source code for their implementation of TCP/IP stack. The solution given below highlights the changes required with the BSD release 4.4 source code with the notations used by IPv4. It addresses issues relevant to IPv6 wherever applicable. All other implementations of TCP/IP have to be updated in the similar manner.

In this document the term "default router" will refer to the customer edge (CE) router that communicates with the provider network. Also the term "intermediate routers" will refer to all the routers apart from the CE routers.

2. Solution for site multihoming

RFC1122[2] made an extensive study related to different aspects of multihoming. Some of the requirements suggested in that document related to UDP and the application layer were avoided for multihomed hosts in a connected network with a single gateway to reach the outside world. This was achieved by the implementation of TCP/IP by making sure that the interface address of an outgoing packet gets selected based on the route to be followed by the destination address. This criterion holds good in a connected environment with a single gateway to reach the outside world. Once more than one gateway comes into play to reach the outside world, either routing table of the entire world has to be brought in or needs some enhancements within the existing system to make things work.

Whenever a customer network gets service from more than one service provider, the customer network can be viewed as having multiple source-id (user-id) space. Each of these IP domain gets connected to different service providers through different routers. So each interface of customer network may have as many global unicast addresses as many service providers it is connected with. Number of routing entries in the routing table will (roughly) become a multiple of IP domains that it supports. Communication between any two hosts within the customer network will follow the traditional routing mechanism. In order to provide multihoming services it is needed that a host computer always forwards packets to the customer edge router associated to the same IP domain while communicating to someone in the outside world. i.e. if the interface of a host computer H receives an IP address ‘addr1’ and ‘addr2’ from two service providers P1 and P2 which are connected through routers R1 and R2 respectively, host H has to forward a packet to R1 while using its IP address as ‘addr1’ in order to send packets to the outside world. So, host computers as well as the intermediate routers have to use default routing based on the source domain of the source address in the IP header.
In order to achieve this, host computers as well as intermediate routers need to have information related to its IP domain (net address/net mask) and the associated default router for all of its IP domains. They need to have a route entry per IP domain for all of its default routers. These information should be uploaded at the system start up time.

Routing of IP packets (in the ip_output module of the hosts and in the ip_forwarding module of the intermediate routers) need to be modified in the following manner.

If destination address of a packet falls outside of its IP domains, it has to be forwarded to the default router based on the domain that the source address belongs to.

If destination address of the IP header falls within any one of its IP domains, usual routing mechanism has to be followed.

If customer network maintains private IP domain, communication using private IP has to be restricted within private IP space.

UDP (or RAW) based servers that need to support multiple clients simultaneously need to respond to a client’s request with the same source address that the client had specified as the destination address. In order to satisfy this, system needs to introduce two system calls along with the existing system calls (i.e. read, write, send, sendto, recv, recvfrom)

```c
ssize_t recvwithdstaddr (int sockfd, char *buf, size_t nbytes, int flags, struct sockaddr *from, socklen_t *fromlen, struct sockaddr *fromcladdr, socklen_t *fromcladdrlen, struct sockaddr *dst, socklen_t *dstlen, struct sockaddr *dstcladdr, socklen_t *dstcladdrlen);

'recvwithdstaddr' receives data with destination address as specified by the sender. It is similar to 'recvfrom' with the additional field 'dst' related to the address of the receiving interface of the host. 'fromcladdr' and 'dstcladdr' will hold the values of co-located care-of addresses (see section 2.2) of source and destination if they happen to be mobile.

ssize_t sendwithsrcaddr (int sockfd, char *buf, size_t nbytes, int flags, struct sockaddr *to, socklen_t tolen, struct sockaddr *dstcladdr, socklen_t dstcladdrlen, struct sockaddr *src, socklen_t srccladdrlen, struct sockaddr *srccladdr, socklen_t srccladdrlen);

'sendwithsrcaddr' sends data specifying the source address of the
outgoing interface of the host. It is similar to 'sendto' with additional parameters related to source address. It behaves like 'sendto' if no address is specified for 'src'. 'srccladdr' and 'dstcladdr' will hold the values of co-located care-of addresses of source and destination.

All the UDP based servers that need to support multiple clients simultaneously, need to replace 'sendto' with 'sendwithsrcaddr' and 'recvfrom' with 'recvwithdstaddr'.

It has been expressed in several documents including RFC4291[3], that a single interface will possess multiple IP addresses in a real IP environment. In these cases, all the UDP servers have to be updated with the system calls 'sendwithsrcaddr' and 'recvwithdstaddr' even if a customer site gets attached to a single gateway to reach the outside world.

The same logic will apply to server applications with RAW sockets. Server applications that are TCP based should work in the usual manner.

2.1. Multihoming and IP Mobility

For a mobile node, its co-located care-of IP address[4] has to be bound to one of the IP addresses supported by the service providers (if mobile node advertises more than one address, the home agent will get confused, also there are other implications). Transport layer must ensure that the 'home address' gets tightly coupled with that particular IP address.

A mobile node in a foreign site will have all the IP addresses supported by the foreign site as well as its "Home Address". As the mobile node will also communicate with the outside world with its "Home Address", user should get a provision to choose its "Home Address" while initiating communication. If mobile node makes use of the address of foreign site for applications that do not need its "Home Address" (say, accessing a web site) cost of communication will get reduced. This feature is useful when a mobile user is in a foreign site but remains within the same sphere of influence (say an user lives in one city but works in a different city which is in a different sphere of influence and likes to access web during his working hours).

If "Home Address" is selected for communication, the transport layer of the mobile node should use its care-of address as the source address and pass its "Home Address" as an option field in the stack. This is because multihoming expects the source address as the deciding factor for packet forwarding.
All the issues that need to be handled for IP mobility have been thoroughly discussed in section 5 of the architectural specification[7].

2.2. Selection of source and destination address

If a source network is connected with ‘n’ service providers and the destination network is connected with ‘m’ service providers, there will be a possible ‘m*n’ combination of source-destination pairs for connection between source and destination. So, application program needs to select a source and destination address before initiating communication with the destination.

A system call needs to be introduced to get the source address based on the destination address. If application program needs to use the destination address directly, it needs to use this system call.

```c
int getcommaddr(int sockfd, struct in_addr *dst, struct addr_pair *endpts);
```

‘addr_pair’ holds the addresses of communication end points as follows:

```c
struct addr_pair {
    struct in_addr src;
    struct in_addr dst;
};
```

‘getcommaddr’ returns the number of source-destination pairs for communication; the field ‘endpt’ will hold the array of these addresses. The array will be in sorted manner based on the best possible route. ‘sockfd’ is used to get the ‘type of service’ assigned. So, an application program needs to set its type of service before using this call.

‘getcommaddr’ needs to call a routine ‘getmappedaddr’[7] to resolve the mapped provider assigned addresses of a provider independent address.

```c
int getmappedaddr(struct in_addr *piaddr, struct in_addr *mpiaddr);
```

‘getmappedaddr’ will return number of mapped addresses and ‘mpiaddr’ will hold their values.

Users may use name instead of IP address to reach the destination. A new system call needs to be introduced ‘gethostbynamewithsrcaddr’, which is an extension to ‘gethostbyname’ as follows:
struct hostent *gethostbynamewithsrcaddr(int sockfd,const char *name, 
    int *nroutes, struct addr_pair *endpts);

'gethostbynamewithsrcaddr' takes 'name' and 'sockfd' as input 
parameters and finds out the best possible route to reach the 
destination. It returns the pointer to the 'hostent' structure as 
returned by 'gethostbyname' system call. The parameter ‘nroutes’ 
gets the number of possible routes to be used and the corresponding 
source and destination addresses gets assigned to ‘endpts’ in sorted 
manner. 'sockfd' is used to get the 'type of service' assigned. So, 
an application program needs to set its type of service before using 
this call.

An application program needs to use these source addresses from the 
top (i.e. the 0th) to establish connection with the destination. It 
needs to bind source address ‘src’ and then connect with the 
destination address ‘dst’.

2.2.1. Path selection

Paths are selected by sending RSVP messages from user to the PE 
routers with the following changes in respective modules.

In order to transport a packet from one network to another, provider 
network sets up a LSP. In RSVP[10,11], resource reservation is 
receiver-initiated. In the Path message, the sending application 
construct Path message using RSVP SENDER_TSPEC and ADSPEC objects. 
The path properties of ADSPEC object gets modified by the network 
elements as the Path message moves from sender to receiver. The 
receiver makes use of SENDER_TSPEC and ADSPEC objects and forms 
FLOWSPEC object and sends back to the network element towards the 
sender. In order to make decision which path an application should 
select from multiple possible paths due to multihoming, composed 
general parameters of ADSPEC object that were received by the 
receiver have to be passed back to the sender by appending them with 
the Resv message.

For best effort service, path is selected based on widest-shortest 
path approach, i.e. the path having the maximum effective available 
bandwidth with minimum NUMBER_OF_IS_HOPS. Effective available 
bandwidth is calculated as

\[
\text{bandwidth allocated to the customer} = \frac{\text{bandwidth allocated to customer}}{\text{gross effective bandwidth allocated to customers}} \times \text{AVAILABLE_PATH_BANDWIDTH}
\]

If (Effective available bandwidth > unused bandwidth 
allocated to the customer)
Effective available bandwidth = unused bandwidth allocated to the customer.

When a Path message is sent from a user to the ingress PE router, for best-effort service the PE router sets up a LSP with the egress PE router and stores the path attributes with the ADSPEC objects if no LSP has already been created. The ingress PE router sends the path attributes (with AVAILABLE_PATH_BANDWIDTH set as Effective available bandwidth) to the sender. If ingress PE router finds an existing LSP for the destination node, it sends the path attributes associated to the LSP.

PE routers need to maintain a list of customers that have accessed the LSP with the last time of access. At the end of each RSVP refresh time, it needs to check the list and delete those entries whose last time of access exceeds the time period of RSVP refresh time. Gross effective bandwidth is calculated as the sum of bandwidths allocated to all the customers available in the list.

The above equation is applicable when communication takes place between global unicast/multicast addresses. In case of VPN, service providers allocate fixed bandwidth path between two customer locations. So, when communication takes place between private addresses actual unused bandwidth of that path has to be returned.

For Guaranteed bandwidth[14] and Controlled-Load service[13] path is selected with MINIMUM_PATH_LATENCY with minimum NUMBER_OF_IS_HOPS, also sender applications need to send PathTear messages for all the paths that are not selected.

A PE router will be in a different address space than the address space of the customer network. As hosts need not be aware of the PE routers, hosts need to send queries to the CE router to get the address of the PE router and store the same in their cache, the way it works with DNS.

2.2.1.1. RSVP extension for path selection from client application

As stated above, for client application to select path, RSVP Resv message needs to pass back composed general parameters of ADSPEC object that were received by the receiver. It is done by appending default ADSPEC general parameters (service 1) NUMBER_OF_IS_HOPS, AVAILABLE_PATH_BANDWIDTH, and MINIMUM_PATH_LATENCY[15] with the FLOWSPEC object. These parameters need to be returned to the ingress PE router without modification.

FLOWSPEC object for Controlled-Load service as defined in RFC 2210[16] will appear to be as follows:
(a) - Message format version number (0)
(b) - Overall length (13 words not including header)
(c) - Service header, service number 5 (Controlled-Load)
(d) - Length of controlled-load data, 12 words not including per-service header
(e) - Parameter ID, parameter 127 (Token Bucket TSpec)
(f) - Parameter 127 flags (none set)
(g) - Parameter 127 length, 5 words not including per-service
(h) - Parameter ID, parameter 4 (Number-of-IS-hops param from [RFC 2215])
(i) - Parameter 4 flag byte
(j) - Parameter 4 length, 1 word not including header
(k) - Parameter ID, parameter 6 (Path-BW param from [RFC 2215])
(l) - Parameter 6 flag byte
(m) - Parameter 6 length, 1 word not including header
(n) - Parameter ID, parameter 8 (minimum path latency from [RFC 2215])
Internet Draft        Solution for Site Multihoming     October 12, 2018

(o) - Parameter 8 flag byte
(p) - Parameter 8 length, 1 word not including header

FLOWSPEC object for Guaranteed bandwidth service as defined in RFC 2210[16] will appear to be as follows:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>23</td>
<td>16</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

512 bytes

(a) - Message format version number (0)
(b) - Overall length (16 words not including header)
(c) - Service header, service number 2 (Guaranteed)
(d) - Length of per-service data, 15 words not including per-service header
(e) - Parameter ID, parameter 127 (Token Bucket TSpec)
(f) - Parameter 127 flags (none set)
(g) - Parameter 127 length, 5 words not including parameter header
(h) - Parameter ID, parameter 130 (Guaranteed Service RSpec)
(i) - Parameter 130 flags (none set)
(j) - Parameter 130 length, 2 words not including parameter header
(k) - Parameter ID, parameter 4 (Number-of-IS-hops param from [RFC 2215])
(l) - Parameter 4 flag byte
(m) - Parameter 4 length, 1 word not including header
(n) - Parameter ID, parameter 6 (Path-BW param from [RFC 2215])
(o) - Parameter 6 flag byte
(p) - Parameter 6 length, 1 word not including header
(q) - Parameter ID, parameter 8 (minimum path latency from [RFC 2215])
(r) - Parameter 8 flag byte
(s) - Parameter 8 length, 1 word not including header

2.2.2. Link failure and switch over to an alternate route

As stated in section 2.1, there are possible "m*n" routes. Client applications select any one of them for communication. If communication fails due to link failure, it may be desirable to switch over to an alternate route (application programs must ensure that it conforms to the requirement of the application).

In reality link failure is a rare phenomenon; so detection of link failure should not become an overhead for the network. Fault gets detected first at the local site where the fault is associated with. Say, if CE-PE link fails, it is the CE router that comes to know about it at the beginning. So, the local site needs to take initiative for the switchover operation. When failure happens, system generates trap which triggers the operation for switchover to an alternate route.

The steps can be summarized as follows:

- When client application calls 'getcommaddr' or 'gethostbynamewithsrcaddr' system finds out a list of possible "source-destination" pairs for communication. If number of routes happen to be more than one rest of the steps are followed.

- Client application establishes a TLS [5] session with its peer after 5 unit tuple gets established. After handshake operation, client application sends the list of source-destination pair to its peer in secured mode. Exchange of routes is required because failure may happen in the remote site too;
o Both client application and its peer store security parameters of TLS session and the list of source-destination routes with the protocol control block (PCB) using ‘setsockopt’ which informs the system to activate switchover operation if there is a link failure;

o When CE router detects failure of CE-PE link, it broadcasts an ICMP message ICMP_LINKFAILURE_CE_PE_LINK to all the hosts.

o On receiving ICMP_LINKFAILURE_CE_PE_LINK, system goes through the list of PCB and gets the list of applications for which it needs to start the switchover operation. For any such particular application, it prepares the list of possible routes for communication through the active links. It tries to set alternate route to its peer by sending ICMP message ICMP_LINKFAILURE_SET_ALT_ROUTE in secured mode with the best possible route.

o On receiving ICMP_LINKFAILURE_SET_ALT_ROUTE, peer host checks whether there is any application in the list of PCB where the request will be applicable. On finding the right PCB, it sets the alternate route and sends a message ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED to its peer.

o On receiving ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED, system sets the alternate route and completes the operation of switchover.

So, it introduces an ICMP message of type ICMP_LINKFAILURE <IANA_TBD1> with the following codes:

<table>
<thead>
<tr>
<th>ICMP_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMP_LINKFAILURE_CE_PE_LINK</td>
</tr>
<tr>
<td>ICMP_LINKFAILURE_CE_FAILURE</td>
</tr>
<tr>
<td>ICMP_LINKFAILURE_SET_ALT_ROUTE</td>
</tr>
<tr>
<td>ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED</td>
</tr>
</tbody>
</table>

In order to provide secured communication it needs to depend on security protocol SSL/TLS. Security parameters e.g. secret key, compression method and cipher spec are stored in the PCB. ICMP messages will have two parts; information in the first part, i.e. ‘struct icmp’ will hold all the necessary information to locate the connection entry in the list of PCB. The second part will hold the information related to the operation and will be in encrypted form with record header. So, changes within a PCB entry is allowed only if ICMP message is received in a secured mode.

It introduces an element ‘struct id_pcb’ inside union ‘icmp_dun’ of ‘struct icmp’ as follows:

```c
struct icmp {
    u_char   icmp_type;   /* type of message, see below */
    u_char   icmp_code;   /* type sub code */
```
u_short icmp_cksum; /* ones complement cksum of struct */
union {
    u_char ih_pptr;  /* ICMP_PARAMPROB */
    struct in_addr ih_gwaddr;  /* ICMP_REDIRECT */
    struct ih_idseq {
        uint16_t icd_id;  /* network format */
        uint16_t icd_seq; /* network format */
    } ih_idseq;
    int ih_void;
} /* ICMP_UNREACH_NEEDFRAG -- Path MTU Discovery (RFC1191) */
union {
    struct ih_pmtu {
        uint16_t ipm_void;  /* network format */
        uint16_t ipm_nextmtu; /* network format */
    } ih_pmtu;
    struct ih_rtradv {
        u_char irt_num_addrs;
        u_char irt_wpa;
        u_int16_t irt_lifetime;
    } ih_rtradv;
} icmp_hun;
union {
    struct id_ts {
        uint32_t its_otime;  /* Originate */
        uint32_t its_rtime; /* Receive */
        uint32_t its_ttime; /* Transmit */
    } id_ts;
    struct id_ip {
        struct ip idi_ip;
    } id_ip;
    struct id_pcb {
        u_char ipcb_ip_proto; /* protocol TCP/UDP */
        struct in_addr ipcb_laddr, /* source address */
        ipcb_faddr; /* destination address */
        u_short ipcb_lport,  /* source port */
        ipcb_fport; /* destination port */
    } id_pcb;
    struct icmp_ra_addr id_radv;
    u_int32_t id_mask;
    char id_data[1];
} icmp_dun;

'struct inpcb' of protocol control block includes four new fields
'inp_lf_n_routes', 'inp_lf_stat', 'inp_lf_routes' and
'inp_seq_params' of type SecParams (SecParams is a type of struct
whose elements are elements of SecurityParameters as defined in
section 6.1 of RFC5246 [5]) as follows:
struct inpcb {
    struct inpcb *inp_next, *inp_prev; /* doubly linked list */
    struct inpcb *inp_head; /* pointer back to chain of inpcb’s for
                             this protocol */
    struct in_addr inp_faddr; /* foreign IP address */
    u_short inp_fport; /* foreign port# */
    struct in_addr inp_laddr; /* local IP address */
    u_short inp_lport; /* local port# */
    struct in_addr inp_fcladdr; /* foreign care-of address */
    struct in_addr inp_lcladdr; /* local care-of address */
    struct in_addr inp_hagentaddr; /* address of home agent */
    struct socket *inp_socket; /* back pointer to socket */
    caddr_t inp_ppcb; /* pointer to per-protocol pcb */
    struct route inp_route; /* placeholder for routing entry */
    int inp_flags; /* generic IP/datagram flags */
    struct ip inp_ip; /* header prototype; should have more */
    struct mbuf *inp_options; /* IP options */
    struct ip_moptions *inp_moptions; /* IP multicast options */
    u_char inp_lf_n_routes; /* number of possible routes */
    u_char inp_lf_stat; /* state of switchover;
                          STAT_DO_NOT_ALTER(0)/STAT_ALTER(1) */
    struct addr_pair *inp_lf_routes; /* pointer to the array of routes*/
    SecParams inp_seq_params; /* security parameters */
};

From application layer, the field ‘inp_seq_params’ is set with the
system call ‘setsockopt’ by introducing a new socket option
SO_SEQPARAM of level SOL_SOCKET; route information i.e.
inp_lf_n_route, inp_lf_routes and inp_lf_stat are set by system call
‘setsockopt’ by introducing another socket option SO_LFROUTES of
level SOL_SOCKET.

setsockopt (sockfd, SOL_SOCKET, SO_SEQPARAM, (char *)&seq_param,
sizeof(SecurityParameters));

setsockopt (sockfd, SOL_SOCKET, SO_LFROUTES, (char *)routes, sizeof
(struct addr_pair)*n_routes);

ICMP messages with ‘icmp_code’ ICMP_LINKFAILURE_SET_ALT_ROUTE and
ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED will have same format as
follows:

Information of the current active link of the PCB entry i.e. protocol
id, source address, destination address, source port and destination
port are set with the fields of ‘struct id_pcb’ of ‘struct icmp’. The
encrypted part of the message will have three fields, source address
and destination address of the alternate route and ICMP code (i.e.
ICMP_LINKFAILURE_SET_ALT_ROUTE/
ICMP_LINKFAILURE_ALT_ROUTES_ESTABLISHED) as it was set with the ICMP header.

Recipient of these messages needs to search PCB entry in the following manner:

If "source port", "destination port" and "protocol id" of incoming ICMP message matches with any entry in the list of PCB and the fields "source address" and "destination address" of the ICMP message matches with any entry of 'inp_if_routes' of the corresponding entry in the PCB, it will be considered as a match. If no matching entry is found, message has to be dropped. With the security information of the PCB entry, the encrypted part of the ICMP message gets decrypted. If it fails to decrypt the message or the message received with invalid MAC, message needs to be dropped. If ICMP code in the header does not match with that of the encrypted part, the message also needs to be dropped.

Details of the ICMP operations are described below:

ICMP_LINKFAILURE_CE_PE_LINK

CE router detects link failure and sends this message to all the users in the network; The field ‘icmp_gwadd’ of ‘struct icmp’ holds the IP address of the PE router.

ICMP_LINKFAILURE_CE_FAILURE

CE router itself may fail. It gets detected by alternate CE router. CE routers send keep-alive messages between themselves at regular interval to detect this failure. The field ‘icmp_gwadd’ of ‘struct icmp’ holds the IP address of the faulty CE router.

ICMP_LINKFAILURE_SET_ALT_ROUTE

This message is sent by a host after receiving ICMP broadcast message ICMP_LINKFAILURE_CE_PE_LINK or ICMP_LINKFAILURE_CE_FAILURE from a CE router for all the entries of PCB whose ('inp_if_stat' = STAT_ALTER and source-destination route passes through the failed link), to their peer. It maintains a list of information where each entry will have the connection details including the best possible route. For best effort traffic route is selected by sending echo messages and calculating round trip delay; for the rest it follows the approach stated in section 2.1.1. For each entry in the list, host sends ICMP_LINKFAILURE_SET_ALT_ROUTE for (arbitrary) ‘n’ number of times with an (arbitrary) interval of ‘t’ msecs (sufficient enough for the reply of ICMP_LINKFAILURE_SET_ALT_ROUTE comes back and gets processed; roughly twice the round trip delay) till it receives a
positive acknowledgment ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED from its peer. On receiving a positive acknowledgment ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED, it deletes the corresponding entry from the list and updates the route information in the PCB.

ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED

On receiving ICMP_LINKFAILURE_SET_ALT_ROUTE, host needs to look for a match in the PCB. If there is a match, host sends ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED to its peer on successful completion of changing 'source address' and 'destination address' with the desired value of the alternate route in the PCB. The message will contain all the fields as that of the receiving message by setting 'icmp_code' as ICMP_LINKFAILURE_ATL_ROUTE_ESTABLISHED both at the header part as well as at the encrypted part.

Switchover operation requires some amount of time. This duration is under the tolerance limit for best effort traffic. For Guaranteed bandwidth and Controlled-Load service as the circuit needs to be reestablished, it may cause flicker. This situation can be avoided by maintaining back-up circuit through an alternate route. As link failure is a rare phenomenon, this feature can be provided on on-demand basis or based on the application type.

2.3. Implementation aspects

Following changes are expected with the source code of BSD.

Introduce ip_domain structure and some parameters as follows:

```c
struct ip_domain {
    struct in_addr net_addr;
    struct in_addr net_mask;
    struct in_addr def_router;
};
#define MAX_IP_DOMAINS 16
short num_ipdomains;
struct ip_domain *ipdomain[MAX_IP_DOMAINS];
```

If customer network maintains private IP domain (along with the user-id space provided by the service providers) and expects its communication to be confined within its own space, 'def_router' has to be set as NULL.

Upload IP domain information for all of its IP domains during system start up. These domain information can be uploaded through router advertisement or through DHCP. The domain information should contain the next hop address to reach the corresponding default router as
There has to be a provision to upload these information through
'sysctl' to configure them manually.

Three new 'sysctl' routines have to be introduced under the 'ip' node
of the MIB tree (i.e. under CTL_NET, PF_INET, IPPROTO_IP)
IPCTL_NUM_DOMAINS, IPCTL_DOMAIN and IPCTL_USE_HOMEADDR (applicable
for mobile node). Both IPCTL_NUM_DOMAINS and IPCTL_USE_HOMEADDR are
of type CTLTYPE_INT and IPCTL_DOMAIN is of type CTLTYPE_NODE. Using
'sysctl' IPCTL_NUM_DOMAINS has to be configured first. Configuration
of IPCTL_NUM_DOMAINS has to populate IPCTL_NUM_DOMAIN entries of
nodes under IPCTL_DOMAIN and for each of these nodes three MIB
attributes DOMAIN_NET_ADDR, DOMAIN_NET_MASK and DOMAIN_DEF_ROUTER
(each of type CTLTYPE_NODE) has to be allocated.

All the routers as well as hosts that are having interfaces
connecting to multiple subnets need to be configured through
'sysctl'.

Mobile users should get provision to change IPCTL_USE_HOMEADDR
attribute dynamically.

Add a route entry for all the default routers during system start up.

2.3.1. Processing of system call 'getcommaddr'

Introduce a routine (say 'getendpointaddr') that will find out a list
of source-destination addresses sorted in order based on sending
Path messages between a list of source addresses to a list of
destination addresses. The routine should select the service type
based on the type of service field (which can be obtained by calling
'getsockopt' with the socket id 'sockfd' passed as a parameter).

System call 'getcommaddr' has to be processed in the following
manner:

If destination address of the IP packet falls outside of its
IP domains {
    If destination address is from private address space {
        if the host is having only one interface {
            for each private address assigned to the interface get
            an entry for the source list.
        }
    }
    else {
        for all the default routers {
            use 'rtalloc' to get the next hop address for the
default router.
        }
    }
}

If destination address of the IP packet falls inside of its
IP domains {
    use 'rtalloc' to get the next hop address for the
default router.
}

else {
    use 'rtalloc' to get the next hop address for the
default router.
get an entry for the source list based on
the outgoing interface ‘ia’, and the private
address associated with the default router.
}
}
destination list will have a lone entry with the
destination private address.
}
else {
    If destination address is provider independent {
        call ’getmappedaddr’ to get all the associated PA addresses;
        for each PA address get an entry of the destination list
    } else {
        get a lone entry for the destination list with the
destination address.
    }
}

If user has selected its "Home Address" {
    /*Applicable to IP mobility*/
    get a lone entry in the source list with the "Home Address".
} else {
    if the host is having only one interface {
        for each global unicast address of the interface,
        get an entry for the source list.
    } else {
        for all the default routers {
            use ’rtalloc’ to get the next hop address for the
default router.

            get an entry for the source list based on
            the outgoing interface ‘ia’, and the global unicast
            address associated with the default router.
        }
    }
}
}
call ’getendpointaddr’ to get the list of source-destination
addresses in sorted manner.
}
else { /* i.e. destination address is inside its IP domains */
    use ’rtalloc’ to get the next hop address for the
destination address.

    if destination address is a link local address {
        select source address based on the outgoing interface
and the link local address assigned to it.
}
else {
    select source address based on the outgoing interface
    and the domain that the destination address belongs to.
}
there is only one possible source-destination combination.

2.3.2. Processing of ‘gethostbynamewithsrcaddr’

System call ‘gethostbynamewithsrcaddr’ has to be processed in the following manner:

This is an enhancement of the system call ‘gethostbyname’. ‘gethostbyname’ calls three routines that performs host table search, NIS search and DNS search. Once name is resolved, following additions are expected to resolve source-destination pair.

If ‘hostent’ structure contains addresses which are inside its IP domains {
    if ‘hostent’ structure contains a private address {
        Assign destination address as a private address contained in ‘hostent’;

        use ‘rtalloc’ to get the next hop address for the destination address.

        select source address based on the outgoing interface
        and the domain that the destination address belongs to.
    }
    else {
        Select a global unicast address contained in ‘hostent’ for destination address.

        use ‘rtalloc’ to get the next hop address for the destination address.

        select source address based on the outgoing interface
        and the domain that the destination address belongs to.
    }
    there is only one possible source-destination combination.
}
else {
    if ‘hostent’ structure contains private address {
        if host is having only one interface {
            for each private address assigned to the interface get an entry for the source list.
else {
    for all the default routers {
        use 'rtaalloc' to get the next hop address for the default router.
        get an entry for the source list based on the outgoing interface 'ia', and the private address associated with the default router.
    }
}
for each private address in the 'hostent' structure
get an entry for the destination list.
}
else {
    for each PA address in the 'hostent' structure
get an entry for the destination list.
    if user has selected its "Home Address" {
        /*Applicable only to IP mobility */
        get a lone entry in the source list with the "Home Address".
    }
    else {
        if the host is having only one interface {
            for each global unicast address of the interface,
            get an entry for the source list.
        }
        else {
            for all the default routers {
                use 'rtaalloc' to get the next hop address for the default router.
                get an entry for the source list based on the outgoing interface 'ia', and the global unicast address associated with the default router.
            }
        }
    }
}
call 'getendpointaddr' to get the list of source-destination addresses in sorted manner.
}

2.3.3. Changes required in ip_output and ip_forwarding modules

Execute the following steps in the 'ip_output' routine of the IP stack before it calls 'rtaalloc' for route look up.
If destination address of the IP packet falls outside of its IP domains {
    get def router address based on the IP domain the source address belongs to.
    use 'rtalloc' to get the next hop address for the def router.
    Forward the packet to the next hop.
} else { /* i.e. destination address is inside its IP domains */
    follow the usual procedure to forward packets
}

In BSD, the 'ip_forwarding' routine calls 'ip_output'; so it should be left as it is.

2.3.4. Processing of protocol input routines and socket IO system calls

Protocol input routines need to locate the socket/process in the usual manner with the 5 unit tuple (i.e. protocol, source address, source port, destination address, destination port).

When a packet is received by a mobile node (at a foreign site), it can be received in two modes. It can be received directly from the correspondent node with the 'destination address' as the co-located care-of address and its home address in the IP stack (see section 4.1 of RFC6275[8]). In the second mode the packet can be received via the home agent using IP over IP. Once the IP layer receives a packet with IP over IP, it is supposed to strip off the outer header before passing the packet to the protocol input routine. In this case packet will be received by the protocol input routine with destination address as the home address of the mobile node with no information related to its care-of address. So, protocol input routine needs to check whether the destination address of the received packet belongs to any one of its IP domains. If it does not, it needs to find out the co-located care-of address by going through the interface list if it is not already found in the packet received. This information is needed by the TCP input routine while processing a SYN message. It is also needed by the UDP/RAW modules while processing the system call 'recvwithdstaddr'.

While processing the output routines like 'sendwithsrcaddr', 'sendto', UDP/RAW modules needs to check the parameters related to source address, source port, destination address, destination port, care-of address of the source, care-of address of the destination in the protocol control block. Parameters in the PCB should prevail over parameters passed by the system call while forming the IP packet.
2.4. Multihoming and VPN

For a corporate that maintains multiple offices and communicates within themselves through private address space using VPN needs to distribute its entire private address space to all its site in a suitable manner. Each one of its offices will get multiple private address space where each of them will be associated with a particular link. Let us consider one of its offices gets connected to two providers P1 and P2 and gets address space as 'unicastNetAddr1'/unicastNetMask1' and 'unicastNetAddr2'/unicastNetMask2' respectively. It also gets assigned private address space from its corporate as 'privateDomainNetAddr1'/privateDomainNetMask1' and 'privateDomainNetAddr2'/privateDomainNetMask2' which will be associated with the CE routers CE1 and CE2 respectively.

All hosts as well as the intermediate routers will have four entries of ip_domain:

1: 'net_addr' = 'unicastNetAddr1'
   'net_mask' = 'unicastNetMask1'
   'def_router' = CE1
2: 'net_addr' = 'unicastNetAddr2'
   'net_mask' = 'unicastNetMask2'
   'def_router' = CE2
3: 'net_addr' = 'privateDomainNetAddr1'
   'net_mask' = 'privateDomainNetMask1'
   'def_router' = CE1
4: 'net_addr' = 'privateDomainNetAddr2'
   'net_mask' = 'privateDomainNetMask2'
   'def_router' = CE2

2.5. IP Address Stacking

IP address stacking in IPv6 is performed with the approach introduced in section 6.4 of RFC6275[8] with slight modification. RFC6275 describes how to pass "Home Address" as well as co-located care-of address of the destination address if it happen to be mobile. The same approach has been extended to support IP address stacking for the source address and to support IP address stacking for both source address as well as destination address. The "Reserved" space in the type 2 routing header has been split into two parts; an one octet field to address the "Stacking Type" and the rest 3 octets are left as Reserved.

Stacking Type is interpreted as follows:

Stacking Type=0
Source Address: Address of the sender.
Destination Address: co-located care-of address of the receiver.
Address 1: Home Address of the receiver.
Hdr Ext Len=2.

So, type 2 routing header for stacking type 0 will be as follows:

```
+---------------------------------------------+
| Next Header | Hdr Ext Len=2 | Routing Type=2 | Segments Left=1 |
+---------------------------------------------+
| Stacking Type=0 | Reserved |                          |
+---------------------------------------------+
| Address 1: Home Address of the receiver |
| + | |
+---------------------------------------------+
```

Stacking Type=1
Source Address: co-located care-of address of the sender.
Destination address: Address of the receiver.
Address 1: Home Address of the sender.
Hdr Ext Len=2.

So, type 2 routing header for stacking type 1 will be as follows:

```
+---------------------------------------------+
| Next Header | Hdr Ext Len=2 | Routing Type=2 | Segments Left=1 |
+---------------------------------------------+
| Stacking Type=1 | Reserved |                          |
+---------------------------------------------+
| Address 1: Home Address of the sender |
| + | |
+---------------------------------------------+
```

Stacking Type 2
Source Address: co-located care-of address of the sender.
Destination Address: co-located care-of address of the receiver.
Address 1: Home Address of the sender.
Address 2: Home Address of the receiver.
Hdr Ext Len=4.

So, type 2 routing header for stacking type 2 will be as follows:

```
++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  | Hdr Ext Len=4 | Routing Type=2|Segments Left=1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Stacking Type=2|                Reserved                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         +
                         |
                         +
                         |
                         Address 1:Home Address of the sender  +
                         |
                         +
                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         +
                         |
                         +
                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         +
                         |
                         +
                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         +
                         |
                         +
                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Next Header**

8-bit selector. Identifies the type of header immediately following the routing header. Uses the same values as the IPv6 Next Header field [9].

**Hdr Ext Len**

4 (8-bit unsigned integer); length of the routing header in 8-octet units, not including the first 8 octets.

**Routing Type**

2 (8-bit unsigned integer).

**Segments Left**

1 (8-bit unsigned integer).

**Stacking Type**

2 (8-bit unsigned integer).

**Reserved**

24-bit reserved field. The value MUST be initialized to zero by the sender, and MUST be ignored by the receiver.
Address 1
Home Address of the sender.

Address 2
Home Address of the receiver.

IP address stacking in IPv4 is performed by introducing new IP option under the option class "Datagram or Network Control", i.e. 0. The option number is 16. The CODE(144) field is followed by one octet field "Stacking Type" followed by two octet reserved space (NULL) as padding followed by the address fields based on the Stacking Type.

Stacking Type is interpreted as follows:
Stacking Type=0
Source Address: Address of the sender.
Destination Address: co-located care-of address of the receiver.
Address 1: Home Address of the receiver.
Header Length:7

Format of IP address stacking option with stacking type 0 in the IP header will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| CODE(144)   |Stacking Type=0| Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+ Address 1:Home Address of the receiver +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Stacking Type=1
Source Address: co-located care-of address of the sender.
Destination Address: Address of the receiver.
Address 1: Home Address of the sender.
Header Length:7

Format of IP address stacking option with stacking type 1 in the IP header will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| CODE(144)   |Stacking Type=1| Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+ Address 1:Home Address of the sender +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Stacking Type=2
Source Address: co-located care-of address of the sender.
Destination Address: co-located care-of address of the receiver.
Address 1: Home Address of the sender.
Address 2: Home Address of the receiver.
Header Length:8

Format of IP address stacking option with stacking type 2 in the IP header will be as follows:

```
+-----------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| CODE(144)       |Stacking Type=2| Reserved                      |
+-----------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  |Address 1:Home Address of the sender|
   +-----------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  |Address 2:Home Address of the receiver|
+-----------------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3. Security Consideration

This document provides a solution for site multihoming of stub networks. Message exchange between source and destination related to link failure has to be done in secured mode as explained in section 2.1.2. For common security related issues that any site may experience, one needs to consult with the "Site Security Handbook", RFC2196[6]. For issues related to IP Mobility, section 5 of RFC5944[4] has to be consulted.

4. IANA Consideration

IANA has assigned an ICMP type <IANA_TBD1> (ICMP_LINKFAILURE) for link failure. IANA has also assigned two socket options SO_SEQPARAM <IANA_TBD2> for security parameters and SO_LFROUTES <IANA_TBD3> for routes to be considered on link failure.

5. Normative References


6. Informative References


7. Author's Address

Shyamaprasad Bandyopadhyay
HL No 205/157/7, Kharagpur 721305, India
Phone: +91 3222 225137
Email: shyamb66@gmail.com